ARM Generic Timer Specification

ARM Architecture Group

Document number:	Derived from: **PRD03-GENC-009660 9.0**
Date of Issue:	22 October 2010
Author:	ARM limited
Authorised by:	

Note

The term ARM is also used to refer to versions of the ARM architecture, for example ARMv6 refers to version 6 of the ARM architecture. The context makes it clear when the term is used in this way.

Contents

1 ABOUT THIS DOCUMENT

The material in this document will be incorporated in the next release of the ARMv7-AR Architecture Reference manual (Revision C).

It is at a Beta specification state. Please communicate any errata or issues to errata@arm.com.

1.1 Change history

This document is derived from an ARM internal document, PRD03-GENC-009660 9.0, and is supplied as PRELIMINARY INFORMATION about the ARM Architecture Generic Timer announced in August 2010.

1.2 References

This document refers to the following documents.

Ref	Doc No	Author(s)	Title
1	PRD03-GENC-008353	ARM Limited	Virtualization Extensions Architecture

2 INTRODUCTION

This specification has been developed in close consultation with a key set of Operating System Vendor (OSV) and silicon partners. The document includes:

- a brief summary of the requirements
- an architecture for the system level counter and timer
- information on the expected system design implications.

The specification is designed to be suitable for use in a system with and without Virtualization. For information on the ARMv7-A Virtualization Extensions see [1].

3 OVERVIEW OF TIMER REQUIREMENTS

3.1 Introduction

Two distinct concepts are covered in the subject of timers:

1. Counting the passing of time. For this document, the object responsible for this functionality is referred to as the *Counter*
2. Scheduling the triggering of events at some point in the future. A number of such events may need to be scheduled, and an object that is responsible for the scheduling of an event is referred to a *Timer.*

This specification covers both a standard counter and a standard timer definition.

3.2 Counter

The key requirements for an architected counter are the following:

- It should run at a constant clock frequency, regardless of the power and clocking state of the processor cores using it. A lower bound on the clock frequency of the counter is in the range 1-10MHz.
    - When the system is running with an inherently low clock frequency, it is acceptable that the precision of the counter is reduced such that the counter is incremented with multiple ticks on a lower frequency. For example, a 10 MHz counter could be updated at either:
        - 1 tick at 10 MHz
        - 500 ticks at 20 KHz

      Such a system, switching between the lower frequency/low precision and high frequency/high precision, should not inject any drift into the count other than the natural drift of the source clocks.

      Note: For system synchronisation reasons, there can be advantages if the ratio between the frequencies is an integer power of 2.
- The counter should continue to run in all levels of powerdown other than completely turning off the system/SoC.
- The size of the counter is recommended to be 64-bits to avoid any need to be concerned about roll-over issues (64bits at 50MHz rolls over in 11.7 thousand years).
- The Counter should start from 0.
- All CPUs in the system must be able to see a uniform view of time. One implementation of this would be that every CPU reads a common counter. The strict definition of the requirement is that it must be impossible for time to be perceived as going backwards if a CPU reads the counter, and then communicates with another agent, which then also reads the counter.
- The read-time of the counter should be fast, and reliably so.
    - This translates into read-times of 10-100nS when the CPU is running at full speed.
    - No situation should cause a delay to the read.
- A single 64-bit read mechanism of the 64-bit counter is preferred.
- The counter is rarely written; making the counter writes memory mapped is acceptable.
- It should be configurable whether the user level of privilege is able to read time directly
- In a system supporting virtualization, the Hypervisor should be able to apply an offset to the counter value that is read within a virtual machine, to allow support for Virtual Time.
- When the system is halted as result of debug, there should be a configurable option to halt the counter.

3.3 Timers

The key requirement is to support a set of timers that provide the ability to schedule events based on the passage of time. Two distinct usage models are:

- Timers that are based on comparison with the counter value. Such comparators would generate an interrupt when:

  ```
  Counter >= CompareValue
  ```

  In this case, CompareValue is a 64-bit unsigned number.

  The use of greater than or equal to is essential for such a timer architecture.

- Timers that count down to 0 (and potentially keep counting down, so that the elapsed time between the required interrupt time and the actual handling can be established).

  Such a timer would fire the interrupt when

  ```
  TimerValue <= 0
  ```

  In this system, TimerValue is a 32-bit signed number

The ARM Timer architecture supports both approaches in a single circuit.

Other requirements are:

- Timers should run at the same clock frequency as the counter, and should be invariant with core clock frequency with the same constraints as the counter.
- Timer read/write times should be fast – order of the granule of a counter tick when the CPU is running at its maximum frequency – and have predictable access.
- Each CPU has its own timer facilities, and these have private access by a CPU. This allows the writing of a timer for the current CPU without a need to understand which CPU is being used.
- Each event from the timers should be routable as a distinct interrupt, including using the FIQ exception (or some other dedicated interrupt for the timer).
- The events caused by timers should be visible to the system beyond the core to allow timer generated events to be used in a system context.
- Timers should continue working in most low-power modes of operation, though the very deepest levels of CPU powerdown can also affect the timers for that CPU.
- At least 1 timer should be provided for each supervisory level of software (OS, Hypervisor and Security).

3.4 Virtual Time

Virtualization support adds the complexity that Physical and Virtual time become separate concepts

- Virtual time measures the elapsed time for an instance of a virtual machine.
- Physical time measures the absolute elapsed time, regardless of which virtual machine is operating

Where virtual machines can be scheduled independently on different processors in a multiprocessor system, only physical time has a meaning between the different processors.

Where a system supports virtualization, it might be necessary to schedule two events in the future – one in virtual time, and one in physical time. In a system without virtualization, the order of these events might be that the virtual time event is earlier than the physical time event. Once virtualization has been deployed, the order of these events might be reversed.

Separate timers are provided to allow Physical time and Virtual time to be independently programmed. In a system that does not support virtualization, these two timers are effectively equivalent. The effect of virtualization on the two timers is, however, different:

- The virtual timer event is calculated with respect to the change of the virtual time of the system
- The physical timer is calculated with respect to the change of the physical time in the system

3.5 Event Stream generation

The ARM architecture includes a power saving concept, Wait For Event, which is used to suspend execution of the processor during polling loops if there is no change to the variable being polled. The possibility of a change to the variable is signalled using an event.

With the architecting of a timer solution, this Wait For Event mechanism can be extended to be used in a situation where it is desired to have a time-out on such polling. Such an approach is often used by a user-level thread to poll a variable for a period of time, and if there is continued failure of the poll, to call the operating system to deschedule the thread of execution. For Wait For Event to use this approach, it is useful for a stream of events (the *Event Stream*) to be generated that periodically wake up the suspended execution to allow checking against a time-out as well as re-polling.

In addition, the ability to generate an Event Stream is desirable to act as a “watchdog” against programming errors, ensuring that erroneous suspension will be woken within a configured time.

Therefore, it is a requirement that each processor is able to specify an Event Stream, for that processor only, using a 4 bit field to select the bit number of the counter used to generate the Event Stream. In addition, the direction of transition of the bit of the counter selected can be specified – this feature helps spread the time at which the events are generated to prevent waking all suspended processors at the same time.

The Event Stream can be disabled, and is disabled from reset. An Event Stream can independently be specified by the Kernel using the Virtual Counter and by the Hypervisor using the Physical Counter.

4 GENERIC TIMER ARCHITECTURE

4.1 Architecture overview

The requirements in section 0 strongly suggest that the architecture for a counter and timers cannot only be implemented in memory mapped space. As a result, the basic form of the ARM Generic Timer is:

- A counter of at least 56-bits located in the system running at a fixed frequency, typically in the range of 1-50MHz. A facility is provided to allow this to be incremented at larger increments, but at a lower frequency, when the system is in low power operation, and transition from high-frequency high-precision to low-frequency-low precision, and back again, might be provided in this counter.

  The mimimum roll-over time for the counter must be 40 years.

  Any 64-bit access to the counter sees the count 0-extended to a 64-bit quantity.

  The counter is required to have an accuracy that it does not gain or lose more than a second in a 24 hour period.
- This counter is held in the system's "Always on" power-domain, and can be accessed in a memory mapped manner for basic control (including clock frequency switching).
- The Count from the counter is distributed to all CPUs in the system using a dedicated mechanism, and each CPU has a counter input – how this is achieved is IMPLEMENTATION DEFINED.
  - A simple Gray-count based scheme for distribution is discussed in section 7
- The frequency of the Counter can be read using a 32-bit CP15 register read within each CPU. The frequency value can be written by Secure Privileged software to allow the frequency of the clock to be programmed into the CP15 register at boot time of the system. The CNTFRQ register is reset to 0.
- The Physical Count can be read using a 64-bit CP15 register read within each CPU
  - User access to the Physical Count is controlled using a Control register per CPU accessed by the Kernel modes
  - Non-secure Kernel access to the Physical Count register is controlled using a Control register per CPU accessed by Hyp mode
- A Virtual Count can be read using a 64-bit CP15 register read, which subtracts a 64-bit Virtual Offset from the Physical count
  - Virtual Offset is held in a 64-bit CP15 register per CPU accessed in Hyp mode
  - User access to the Virtual Count is controlled using a Control register per CPU accessed by the Kernel modes
- The Kernel can specify that an Event Stream for this processor can be generated on the transition of any specified one of the lowest 16 bits of the Virtual Count. The Event Stream specification has :
  - an enable (EVNTEN),
  - a 4 bit field (EVNTI) to specify the counter bit position to select (one of the bottom 16 bits
  - a direction of the transition (EVNTDIR) – 0 is a rising transition; 1 is a falling transition.

  The functionality is:

  SampleBit = (CNTVCT<EVNTI>) XOR EVNTDIR ;

  NewEvent = SampleBit(time_t) AND NOT (SampleBit(time_t-1);

  EVENT_CPU = OR(OTHER EVENTS IN SYSTEM) OR (NewEvent AND EVNTEN);

  Where EVENT_CPU is the Event that is used for the Event logic for this CPU.
- The Hypervisor can specify that an event stream for this processor can be generated of the transition of any specified one of the lowest 16 bits of the Physical Count. The function and specification of the event stream is the same as for the Kernel Event stream, except that it is based on the Physical count.

- In addition, a set of 4 timers per CPU is defined
    - A physical timer for use in Kernel modes/Secure Privileged modes (CNTP_) Banked
        - This register is banked to allow the secure side to have its own copy
        - Non-secure Kernel access to this Timer controlled using a Hyp Control register
    - A virtual timer for use in Kernel modes (CNTV_) Common
    - A physical timer for use in Hyp mode (CNTHP_) Common

  Each CPU makes the output of each timer as an Output pin to the system, and to a dedicated private peripheral interrupt for that CPU.
- The timer functionality defines a programmable 64-bit value:
    - CompareValue

  The function of the timer is:

  `EventTriggered = ((Counter[63:0] – Offset[63:0])[63:0] –CompareValue[63:0] >= 0)`

  Where Offset is:

  0 for all physical timers

  The VirtualOffset for the virtual timer

  An alternative 32-bit view, called TimerValue, is provided to read or write the CompareValue:

  The TimerValue reads:

  `TimerValue = (CompareValue – (Counter – Offset))[31:0]`

  The TimerValue writes CompareValue with:

  `CompareValue = (Counter - Offset) + SignExtend(TimerValue)`

  Note – all values written and read are in standard 2's complement binary.
- Within each CPU, CP15, Register 14 is allocated for the provision of Counter and Timer registers as shown in the following table. Following standard CP15 rules, all registers are UNK at reset unless otherwise stated.

Name	**Function**	**Accessible by:**	**Trustzone**
CNTFRQ	A 32-bit Read of clock ticks per second	Read: Kernel, Hyp, Secure Priv Configurably User access Write: Secure Priv	Common
CNTPCT	A 64-bit Read of Physical Count	Hyp, Secure Priv Configurably: Kernel, User	Common
CNTVCT	A 64-bit Read of (Physical Count - Virtual Offset)	Kernel, Hyp, Secure Priv Configurably User access	Common
CNTVOFF	A 64-bit Read/Write of a Virtual Offset	Hyp, Monitor (SCR.NS==1)	Banked-NSHyp Only
CNTKCTL	A 32-bit Read/Write control register for: ▪ User accessibility of Physical Count	Kernel, Hyp, Secure Priv	Common

	CNTPCT and CNTFRQ (Bit[0]) ▪ User accessibility of Virtual Count CNTVCT and CNTFRQ (Bit[1]) ▪ EVNTEN (Bit[2]) ▪ EVNTDIR (Bit[3]) ▪ EVNTI (Bits[7:4]) ▪ User accessibility of CNTV timer (Bit[8]) ▪ User accessibility of CNTP timer (Bit[9]) The user accessibility is defined as : 0 –access denied; 1 – access permitted Bits[31:10] : UNK/SBZP; Bits[9,8,2:0] Reset to 0 When user access is denied, attempting access in Non-secure or Secure user modes causes an UNDEFINED Exception.		
CNTHCTL	A 32-bit Read/Write control register for: ▪ NS Kernel or User access of Physical Count CNTPCT (Bit[0]) ▪ NS Kernel or User access of Physical Timer CNTP_CVAL, CNTP_TVAL, CNTP_CTL (Bit[1]) ▪ EVNTEN (Bit[2]) ▪ EVNTDIR (Bit[3]) ▪ EVNTI (Bits[7:4]) The kernel or user accessibility is defined as : 0 –access denied; 1 – access permitted Bits[31:8] : UNK/SBZP; Bit[2] Reset to 0, Bits[1:0] reset to 1. When Kernel or user access is denied, attempted access in Non-secure Kernel or Non-privileged modes causes a Hyp Entry Trap. The Non-privileged UNDEFINED exception has priority over a Hyp Entry Trap.	Hyp, Monitor (SCR.NS==1)	Banked-NSHyp Only
<Name_>CVAL	CompareValue – a 64-bit Read/Write value	CNTHP: Hyp, Monitor (SCR.NS==1) CNTV: Kernel, Hyp, Secure Priv, Configurably user access CNTP (S): Secure Priv (Monitor when SCR.NS==0) CNTP(NS): Hyp, Monitor (SCR.NS==1), Configurably Kernel	CNTHP: Banked-NSHyp only CNTV: Common CNTP: Banked
<Name_>TVAL	TimerValue – a 32-bit Read/Write value		
<Name_>CTL	A 32-bit Read/Write control register which can disable the timer and can mask its interrupt (bit[0] resets to 0) Bit[0] – 1- Enable timer; 0 – disable timer Bit[1] – 1 - Mask Interrupt; 0 – unmask interrupt Bit[2] – Int status before masking. RO : 1-asserted; 0-nonasserted Bits[31:3]: UNK/SBZP If the timer is disabled, interrupts are masked, but the timer value continues to count down. The Timer being disabled allows a greater degree of		

	power saving to be achieved in the Timer logic, as well as masking interrupts.		

The 64-bit registers are accessed atomically using MRRC/MCRR instructions

4.2 Register Encodings

MCR/MRC P15, 0 Rt, C14, C0, 0	CNTFRQ
MRRC P15, 0, Rt1, Rt2, C14	CNTPCT
MRRC P15, 1, Rt1, Rt2, C14	CNTVCT
MCRR/MRRC P15, 4, Rt1, Rt2, C14	CNTVOFF
MCR/MRC P15, 0, Rt,C14, C1, 0	CNTKCTL
MCR/MRC P15, 4, Rt,C14, C1, 0	CNTHCTL
MCRR/MRRC P15, 2, Rt1, Rt2, C14	CNTP_CVAL
MCR/MRC P15, 0, Rt, C14, C2,0	CNTP_TVAL
MCR/MRC P15, 0, Rt, C14, C2,1	CNTP_CTL
MCRR/MRRC P15, 3, Rt1, Rt2, C14	CNTV_CVAL
MCR/MRC P15, 0, Rt, C14, C3,0	CNTV_TVAL
MCR/MRC P15, 0, Rt, C14, C3,1	CNTV_CTL
MCRR/MRRC P15, 6, Rt1, Rt2, C14	CNTHP_CVAL
MCR/MRC P15, 4, Rt, C14, C2,0	CNTHP_TVAL
MCR/MRC P15, 4, Rt, C14, C2,1	CNTHP_CTL

4.3 CPUID Mechanism

4.3.1 c0, Processor Feature Register 1 (ID_PFR1)

Bits[19:16] are defined for Generic Timer Support

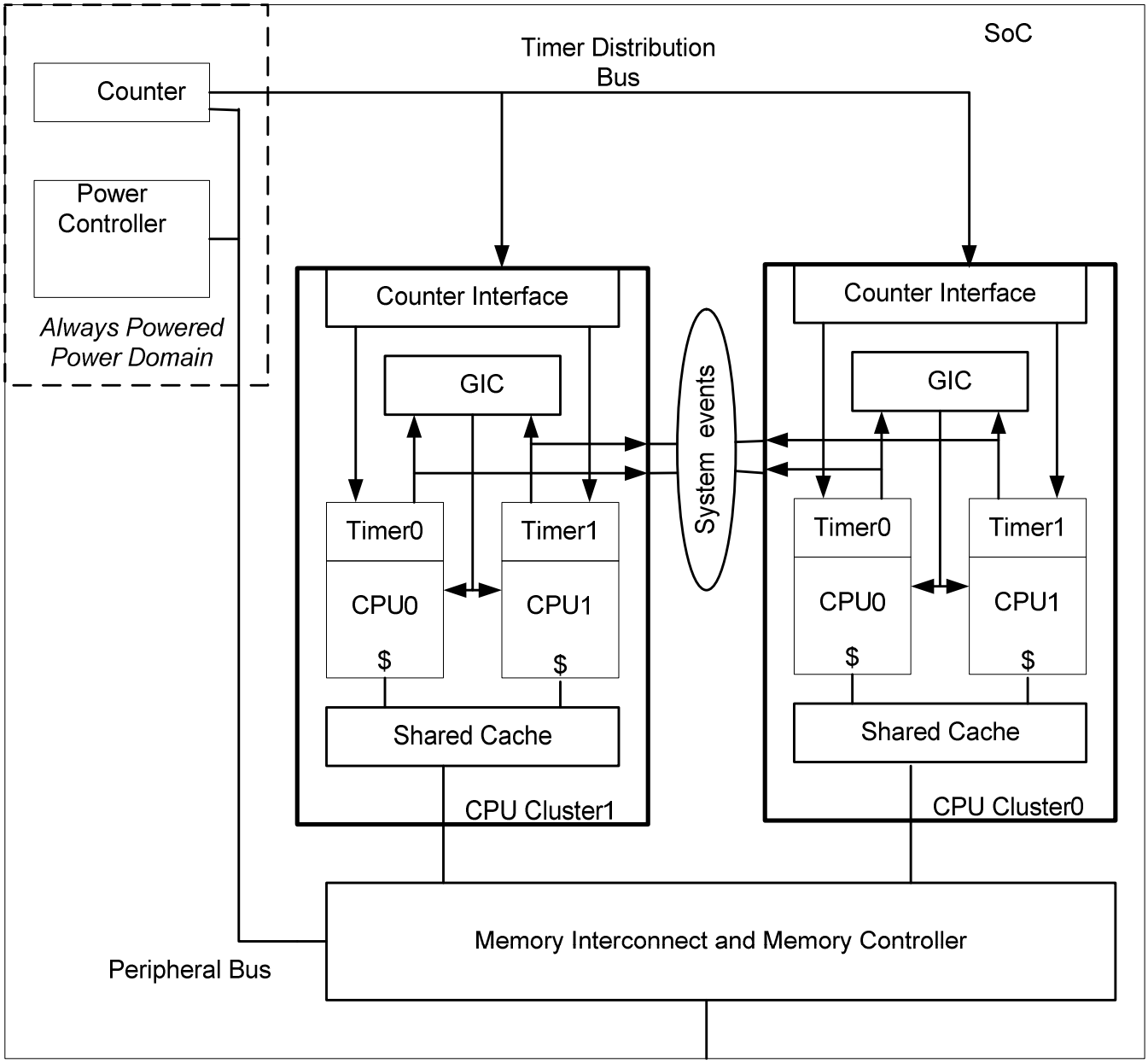
Permitted values are:

0b0000 Not Supported

0b0001 Generic Timer Supported.

4.4 Example System Diagram

The following system diagram shows an example system diagram using the counter architecture described in this document:



5 MEMORY MAPPED COUNTER MODULE

The memory mapped counter module holds the counter in an always powered region within the SoC.

The basic facilities to be provided are:

- Read/Write Control register providing
  - Enable
  - Frequency/Precision change request
  - Halt-on-debug
    - Signal to determine Debug is a pin to the module, derived from cross-trigger or other implementation specific functionality.
  - The write access is rarely performed and should be restricted to Secure side in a system deploying security.
- Writeable Register for writing current count value
  - Writes must occur when the counter is not Enabled or else the effect in UNPREDICTABLE
  - This write is rarely performed and should be restricted to Secure side in a system deploying security.
- Readable Status Register providing:
  - Frequency/Precision change status
- Readable Register holding current count value (64-bit atomic read)
- ID Registers providing:
  - Operating Frequency
  - The increment step of low frequency operation
    - The value 0 reflects that no low-frequency operation is available
  - Standard Primecell ID mechanisms

The mechanism for changing the frequency/precision envisages a request/acknowledge protocol

A write to the control register will select the frequency/precision setting. The hardware will wait until the high frequency count has reached an integral number of low-frequency ticks, and then switch on that cycle to the low-frequency operation. Similarly, a switch from low frequency operation to high frequency operation will occur on a low-frequency clock tick.

The frequency of operation in use is reflected in the Status register, and can be polled by software to ensure that a change has taken place.

In order to provide a number of different lower frequency/lower precision modes, the mechanism for requesting a change of frequency is based around selecting an entry in a Table of Frequency IDs. The first entry in this table is the highest frequency supported by the timer (indicated in ticks per second) and corresponds to the counter incrementing by a single value. Subsequent entries run at a lower frequency – typically these are expected to be at an integer power of 2 divisor of the highest frequency. The first entry in the table that contains 0 indicates the end of the table.

Note: it is acceptable for the table to hold a single non-zero entry to indicate a system where there is no ability to switch frequency.

5.1 Proposed Memory Map

The basic memory mapped counter module requires 2 regions of memory:

- A region for controlling the Counter. This starts from the address CNTControlBase
- A region that allows the current counter value to be read. This starts from the address CNTReadBase

CNTControlBase and CNTReadBase must be at least 4KBytes apart with a 64KB separation recommended. In systems which have Security, CNTControlBase is in Secure memory space only.

The Counter is assumed to be little-endian.

Address	Read/Write	Size	Function
CNTControlBase+0	R/W	32-bit	CNTCR: Counter Control Register:
CNTControlBase+0x4	RO	32-bit	CNTSR: Counter Status Register
CNTControlBase+0x8	R/W	32-bit	CNTCV[31:0]: Counter Count Value Register low order bits
CNTControlBase+0xC	R/W	32-bit	CNTCV[63:32]: Counter Count Value Register high order bits
Table of Frequency IDs			
CNTControlBase+0x20	RO or R/W*	32-bit	CNTFID0: Base Frequency ID
CNTControlBase+0x20+4i	RO or R/W*	32-bit	CNTFIDi: Frequency ID entry i
Unused addresses	RO	32-bit	0
CNTControlBase+0xFD0	RO	12x32-bit	ID Registers
CNTReadBase	RO	32-bit	CNTCV[31:0]: Counter Count Value Register low order bits
CNTReadBase+0x4	RO	32-bit	CNTCV[63:32]: Counter Count Value Register high order bits
Unused addresses	RO	32-bit	0
CNTReadBase+0xFD0	RO	12x32-bit	ID Registers

*For the Frequency ID table, in some systems it might be necessary to be able to program the frequency values that can be selected as part of initial configuration. See section 5.1.1

The ID registers are IMPLEMENTATION DEFINED, though are designed to allow the Primecell ID mechanism to be used if the system architecture requires such a mechanism. The Architectural revision of this version of the Timer specification, as might be reflected in the Primecell ID registers, is 0.

CNTCV can be accessed as an atomic 64-bit quantity in implementations that support a 64-bit atomic access.

CNTCR:

EN (Bit[0]): Enable

0 – Disabled (reset value); Count is not incrementing

1 – Enabled : Count is incrementing

HDBG (Bit[1] Halt On Debug

0 – Debug signal into the Counter has no effect

1 – Debug signal into the Counter halts the Count

UNK/SBZP(Bits[7:2]) RESERVED

FCREQ(Bits[7+N:8]) Request of a Change of Frequency to Frequency value

N bit value that indicates which entry in the frequency table to select

Selecting an unimplemented entry (or the 0 entry) has no effect on the counter

This field is reset to 0.

UNK/SBZP(Bits[31:8+N]) RESERVED

CNTSR:

DBGH(Bit[1]) Debug Halted

FCACK(Bits[7+N:8] Acknowledge of a Change of Frequency

Takes the value of the entry in the frequency table that is currently being used

This field is reset to 0

UNK/SBZP(Bits[31:8+N; 7:2,0]) RESERVED

5.1.1 Ability to Write Frequency ID table

In some systems it might be necessary to be able to program the frequency values that can be selected as part of initial configuration of the system from Power on Reset, in order to allow the Frequency values to be determined within the entire system of the design. The mechanism for such initial system configuration is outside the scope of this architecture, but might involve making the Frequency ID table writeable in the most secure configuration of the system.

Other than in the initial configuration of the Frequency ID information, it is expected that the values in the Frequency ID table will not change, and in some implementations where such configurability is not required, the Frequency ID table will be Read-only.

6 RELATIONSHIP WITH THE CORESIGHT TIMESTAMP COUNTER

The Coresight Timestamp counter has similar requirements to the Generic counter discussed in this document and so it is permissible that they can be combined in systems incorporating both the Coresight Timestamp counter and the Generic Counter. This might mean that the reported frequency of the Generic Counter is significantly higher than the 1-50MHz suggested as a typical frequency of operation. Where the counter is shared with Coresight Timestamp support, the standard increment for the generic timer of the Counter can be greater than 1.

Note: the Coresight Timestamp counter typically runs at a higher clock frequency than is required for the Generic Counter, and as such the appropriate distribution scheme might be determined by the Coresight Timestamp mechanism.

MEMORY MAPPED VIEW OF THE TIMER

In order to allow the functionality of the Timer to be provided for CPUs and other programmable components developed without coprocessor access to the Timer, the following memory map is proposed for a memory mapped timer component placed close to such CPUs. A copy of this structure should be instantiated for each processor or agent using the counter.

Note: this mechanism is described to work with legacy cores. In new implementations of the ARM architecture, the generic timer architecture should be incorporated in a coprocessor mapped manner.

The memory map consists of five Base Addresses, separated by at least 4Kbytes, 64KB recommended. These Base Addresses can be used by the memory management system to provide the same protection of the resources of the timer as the coprocessor mapped instructions deliver.

- CNTSKernelBase — This should be located in the Secure physical memory map. In the Secure translation tables, this should not be user accessible
- CNTSUserBase — This should be located in the Secure physical memory map. In the Secure translation tables, this should be user accessible
- CNTNSHypBase — This should be located in the Non-secure physical memory map. In the Non-secure Hyp Stage 1 translation tables, this should accessible from Hyp mode. In the Secure translation tables, this should accessible from Secure Kernel mode.
- CNTNSKernelBase — This should be located in the Non-secure physical memory map. In the Stage 2 translation tables, this should be accessible from Kernel modes. In the Non-Secure Kernel Stage1 translation tables, this should not be user accessible. In the Non-secure Hyp Stage 1 translation tables, this should accessible from Hyp mode. In the Secure translation tables, this should accessible from Secure Kernel mode.
- CNTNSUserBase — This should be located in the Non-secure physical memory map. In the Stage 2 translation tables, this should be accessible from Kernel modes. In the Non-Secure Kernel Stage1 translation tables, this should be user accessible

The Memory Maps for each base address are shown in the following tables.

Address	**Size**	**Register**		**Notes**
CNTSKernelBase+0	32	CNTPCT[31:0]	RO	
CNTSKernelBase+0x4	32	CNTPCT[63:32]	RO	
CNTSKernelBase+0x8	32	CNTVCT[31:0]	RO	
CNTSKernelBase+0xC	32	CNTVCT[63:32]	RO	
CNTSKernelBase+0x10	32	CNTFRQ	R/W	Writeable to allow initial configuration
CNTSKernelBase+0x14	32	CNTKCTL	R/W	
CNTSKernelBase+0x18	32	CNTVOFF[31:0]	R/W	
CNTSKernelBase+0x1C	32	CNTVOFF63:32]	R/W	
CNTSKernelBase+0x20	32	CNTP_CVAL[31:0]	R/W	Secure version of the timer
CNTSKernelBase+0x24	32	CNTP_CVAL[63:32]	R/W	Secure version of the timer
CNTSKernelBase+0x28	32	CNTP_TVAL	R/W	Secure version of the timer
CNTSKernelBase+0x2C	32	CNTP_CTL	R/W	Secure version of the timer

Address	Size	Register		Notes
CNTSUserBase+0	32	CNTPCT[31:0]	RO	Permitted when CNTKCTL[0]==1
CNTSUserBase+0x4	32	CNTPCT[63:32]	RO	Permitted when CNTKCTL[0]==1
CNTSUserBase+0x8	32	CNTVCT[31:0]	RO	Permitted when CNTKCTL[1]==1
CNTSUserBase+0xC	32	CNTVCT[63:32]	RO	Permitted when CNTKCTL[1]==1

Address	Size	Register		Notes
CNTNSHypBase+0	32	CNTPCT[31:0]	RO	
CNTNSHypBase+0x4	32	CNTPCT[63:32]	RO	
CNTNSHypBase+0x8	32	CNTVCT[31:0]	RO	
CNTNSHypBase+0xC	32	CNTVCT[63:32]	RO	
CNTNSHypBase+0x10	32	CNTFRQ	RO	
CNTNSHypBase+0x14	32	CNTHCTL	R/W	
CNTNSHypBase+0x18	32	CNTVOFF[31:0]	R/W	
CNTNSHypBase+0x1C	32	CNTVOFF63:32]	R/W	
CNTNSHypBase+0x20	32	CNTHP_CVAL[31:0]	R/W	
CNTNSHypBase+0x24	32	CNTHP_CVAL[63:32]	R/W	
CNTNSHypBase+0x28	32	CNTHP_TVAL	R/W	
CNTNSHypBase+0x2C	32	CNTHP_CTL	R/W	

Address	Size	Register		Notes
CNTNSKernelBase+0	32	CNTPCT[31:0]	RO	Permitted when CNTHCTL[0]==1
CNTNSKernelBase+0x4	32	CNTPCT[63:32]	RO	Permitted when CNTHCTL[0]==1
CNTNSKernelBase+0x8	32	CNTVCT[31:0]	RO	
CNTNSKernelBase+0xC	32	CNTVCT[63:32]	RO	
CNTNSKernelBase+0x10	32	CNTFRQ	RO	
CNTNSKernelBase+0x14	32	CNTKCTL	R/W	
CNTNSKernelBase+0x18	32	RAZ/WI		
CNTNSKernelBase+0x1C	32	RAZ/WI		
CNTNSKernelBase+0x20	32	CNTP_CVAL[31:0]	R/W	Non-Secure version of the timer Permitted when CNTHCTL[1]==1
CNTNSKernelBase+0x24	32	CNTP_CVAL[63:32]	R/W	Non-Secure version of the timer Permitted when CNTHCTL[1]==1

CNTNSKernelBase+0x28	32	CNTP_TVAL	R/W	Non-Secure version of the timer Permitted when CNTHCTL[1]==1
CNTNSKernelBase+0x2C	32	CNTP_CTL	R/W	Non-Secure version of the timer Permitted when CNTHCTL[1]==1
CNTNSKernelBase+0x30	32	CNTV_CVAL[31:0]	R/W	
CNTNSKernelBase+0x34	32	CNTV_CVAL[63:32]	R/W	
CNTNSKernelBase+0x38	32	CNTV_TVAL	R/W	
CNTNSKernelBase+0x3C	32	CNTV_CTL	R/W	

Address	**Size**	**Register**		**Notes**
CNTNSUserBase+0	32	CNTPCT[31:0]	RO	Permitted when CNTKCTL[0]==1
CNTNSUserBase+0x4	32	CNTPCT[63:32]	RO	Permitted when CNTKCTL[0]==1
CNTNSUserBase+0x8	32	CNTVCT[31:0]	RO	Permitted when CNTKCTL[1]==1
CNTNSUserBase+0xC	32	CNTVCT[63:32]	RO	Permitted when CNTKCTL[1]==1

Registers to which access is not permitted are RAZ/WI

CNTPCT, CNTVCT, CNTV_CVAL, CNTP_CVAL and CNTHP_CVAL can each be accessed as an atomic 64-bit quantity in implementations that support a 64-bit atomic access.

7 GRAY COUNT TIMER DISTRIBUTION SCHEME

The distribution of the Counter value using a Gray-code provides a relatively simple mechanism to avoid any danger of the count being sampled with an intermediate value even if the clocking is asynchronous. It has a further advantage that the distribution is relatively low power, since only 1 bit changes on the main distribution wires for each clock tick.

A suitable Gray-Coding scheme can be achieved with the following logic:

Gray[N] = Count[N]

Gray[i] = (XOR(Gray[N:i+1])) XOR Count[i] for N-1 >= i >= 0

Count[i] = XOR(Gray[N:i]) for N >= i >= 0

for an N+1 bit counter, where Count is a conventional binary count value, and Gray is the corresponding Gray count value.

This scheme has the advantage of being relatively simple to switch between a low-frequency/low precision operation and a high-frequency high-precision operation (or vice versa), where the ratio of the frequencies is 2^n (n is an integer). Such a switch-over can only occur on the 2^{n+1} boundary to avoid losing the Gray-coding property on a switch-over.